

Atlas: An Example of Using Aspects to Express Architectural Configurations

Gail C. Murphy
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver Canada V6T 1Z4
murphy@cs.ubc.ca

Mik Kersten
Xerox PARC
3333 Coyote Hill Road
Palo Alto CA 94304

mkersten@parc.xerox.com

ABSTRACT

The Advanced Teaching and Learning Academic Server (Atlas) is a software system that supports web-based learning. Students can register for courses, and can navigate through personalized views of course material. Atlas can be dynamically reconfigured to run in various architectural configurations suitable for different network contexts. We expressed these different configurations using aspects; specifically we built Atlas using Xerox PARC's AspectJ, an aspect-oriented programming extension to Java. Aspects were helpful in expressing the different configurations in a modular way. The use of aspects to express the different configurations has made the code easier to debug and easier to read compared to a purely object-oriented approach.

INTRODUCTION

Every month, we see the introduction of more devices with connectivity. Every day, we see an increase in the number of people connected through these different devices. With this increase in connectivity through different devices comes a demand for increasingly flexible applications, such as news clippers that work on both desktop and palm computers.

One way of meeting this demand is to create completely separate applications that work within the different computing environments provided by the different devices. A problem with this approach is that it may mean that an organization has to maintain many different code bases to provide essentially the same functionality. A better solution is to improve the flexibility of the code base to support different architectural configurations in which the application may need to execute.

When developing the Advanced Teaching and Learning Academic Server (Atlas), we faced this requirement of a need for multiple architectural configurations. The Atlas system supports web-based learning: students can register for courses and can navigate through personalized views of course material. Atlas is built to support concurrent access by a few students or by thousands of students. The bandwidth of the connection a student has to an

Atlas installation varies and thus it was desirable to be able to configure Atlas to run in one of four network contexts.

1. *Single server* context. In this context, requests to Atlas are run in different threads on the one server. This context is suitable for a modest number of users.
2. *Application server* context. Two servers are used in this context: a web server and an application server. The web server forwards application requests to the application server. This configuration helps support higher user loads.
3. *Parallel application server* context. To exploit the inherent parallelism associated with requests through the web, Atlas is distributed over a number of servers. Requests are load balanced across the application servers. This configuration provides a better response rate under heavy load.
4. *Applet* context. To provide the performance and responsiveness associated with desktop applications, in this context, most of the Atlas functionality runs in the web browser. This context requires a centralized database to ensure that a student with multiple browsers open is provided a consistent view. The database functionality may be run on the web server.

We implemented Atlas using Xerox PARC's AspectJ™ [3], which provides an aspect-oriented extension to Java™. One of the benefits of using aspect-oriented programming [2] for Atlas was that we were able to capture the code needed to support the different architectural configurations through aspects. This expression of the architectural configurations in aspects helped modularize the code associated with these features and enabled Atlas to support the dynamic reconfiguration of a context.

In this short paper, we describe how we expressed the contexts as aspects. Further details on the use of aspect-oriented programming in Atlas are available elsewhere [1]. Further details on AspectJ can be found at www.aspectj.org. Atlas was built using versions 0.2.0beta4 through beta10 of the AspectJ system.

ASPECTS FOR NETWORK CONTEXTS

Atlas is built to Sun Microsystem's Java Servlet specification.¹ As a consequence, one constraint was clear: requests from a student's browser would arrive at a Java web server and would then be delegated by the server to Atlas. The Atlas code would then be responsible for providing application functionality back to the student.

Given this basic constraint, we separated out the basic infrastructure components for the contexts as an object-oriented framework: the Distributed Servlet Broker (DSB). Atlas builds on this framework. Aspects are used to support configuration of contexts and to tailor the behaviour of the Atlas functionality in those contexts. First, we describe the DSB; then we describe how aspects are used to support the various configurations.

The Distributed Server Broker Framework

The DSB framework consists of three main components.

1. The `DSBClient` handles HTTP requests forwarded from the Java web server and determines to which application the requests should be forwarded.
2. The `DSBServer` provides access to the applications being served.
3. The `DSBNexus` provides access to any databases and registries that may be used by an application.

Using Atlas with the DSB framework involves setting up the `DSBServer` to access Atlas, and configuring the appropriate Atlas databases in the `DSBNexus` component. Atlas functionality is accessed through the `AtlasService` class, which implements

the `javax.servlet.Servlet` interface.

Configuring the DSB to run in a particular network context involves instantiating and configuring DSB components and the application (i.e., `AtlasService`) in the appropriate numbers on various nodes. In addition, some functionality within Atlas must also be tailored to address performance concerns. The complexity of configuring a particular network context is proportional to the level of distribution in the context.

The simplest context is the *single server* context in which all components of the DSB framework and the `AtlasService` run on the web server. In the *applet* context, all but one of the components runs on the user's machine: the `DSBNexus` component is the exception: it must be centralized to allow for multiple browser access.

In the *application server* context (shown in Figure 1a), all three DSB components and the `AtlasService` component run on a separate server. This context requires cross-server communication between the `DSBClient` and `DSBServer`. We used ObjectSpace Voyager² to provide transparent access to the distributed Java objects running on different servers.

The *parallel application server context*, which exploits parallelism of HTTP requests, is more complex (Figure 1b). The `DSBNexus` component runs on a separate node. By centralizing this component, consistency is ensured. In this context, new instantiations of the `AtlasService` may be added at runtime, each instantiation runs on a different server. The `DSBClient` is responsible for load balancing between the available application nodes. As before, ObjectSpace Voyager is used to provide

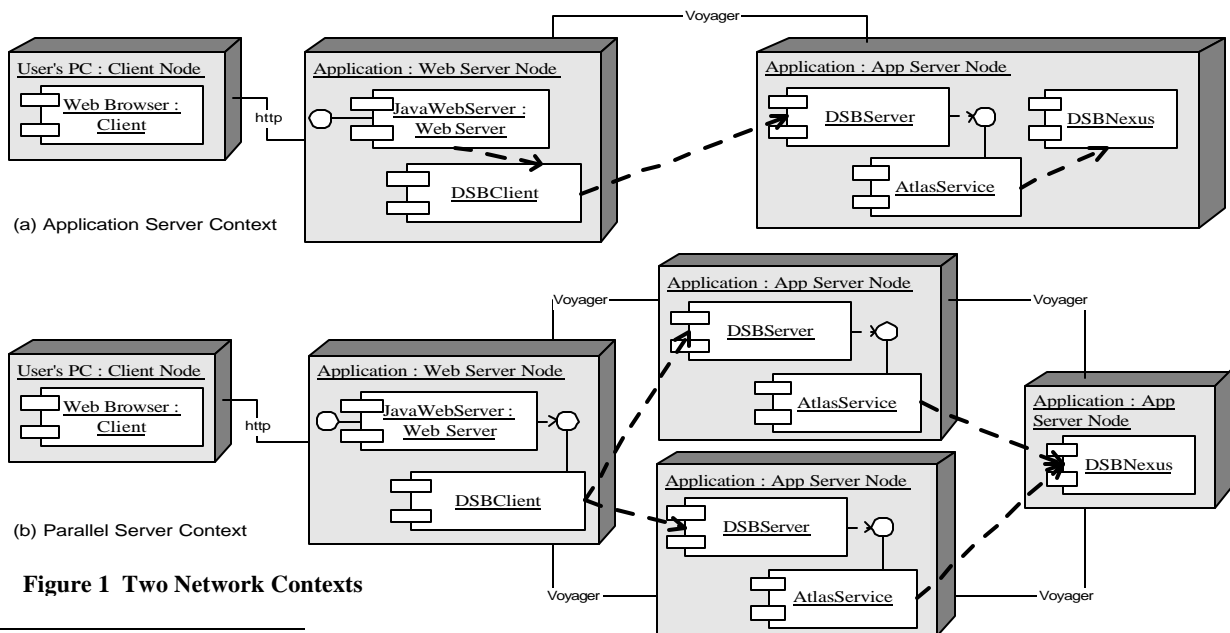


Figure 1 Two Network Contexts

¹ See <http://java.sun.com/products/servlet/index.html>. We used Version 1.2 of the Servlet API.

² See <http://www.objectspace.com/Products/voyager1.htm>.

transparent access to distributed Java objects.

Supporting a particular network context using a purely object-oriented approach would be reasonably straightforward. Supporting all of the network contexts using a purely object-oriented approach would be somewhat more complicated. One possible way to represent the contexts in a purely object-oriented approach would be to introduce classes to represent each context, where each class would implement a particular interface. To instantiate and communicate with DSBServer objects, the DSBClient would delegate through an object representing the appropriate network context. Similarly, the AtlasService would delegate through an appropriate context object to communicate with a DSBNexus object. This approach would require a way to ensure compatible context objects were being used across the system. For instance, you would not want the DSBClient to be acting in an applet context when the AtlasService was acting in a parallel server context.

Aspects for Network Contexts

To eliminate the need to make changes across the system design, we used aspects to support the configuration of the network contexts. Figure 2 depicts the aspects (shown as diamonds) involved in providing the network contexts, and shows how these aspects interact with the source code for the system.³ The NetworkContext aspect generalizes four sub-aspects, each of which provides the appropriate behaviour for a context.

Each of the network context sub-aspects contains the code particular to the associated context. The DefaultContext supports the *single server* context and is the default mode for the application. This aspect alters no behaviour in the application. The AppServer aspect supports the *application server* context, extending the system to handle remote requests between the DSBClient and the DSBServer. The ParallelServer aspect acts similarly but also extends the system to communicate with a remote, centralized database. Both the AppServer and the ParallelServer aspects use the ServerRegistry class to keep track of servers being used. The registry also helps load balance for the parallel server context. The Applet aspect generates a page that contains an applet that acts as a wrapper for a DSBClient. This aspect is also responsible for the additional protocol issues of communicating from an applet rather than a regular node.

The aspects interact with both DSB components and the AtlasService (Figure 2). The AtlasService was written without considering distribution. As a result, a large amount of the code depends on the local execution context. Many classes rely on utilities that cannot execute correctly in a distributed context. In addition, issues, such as concurrency, must be addressed for the

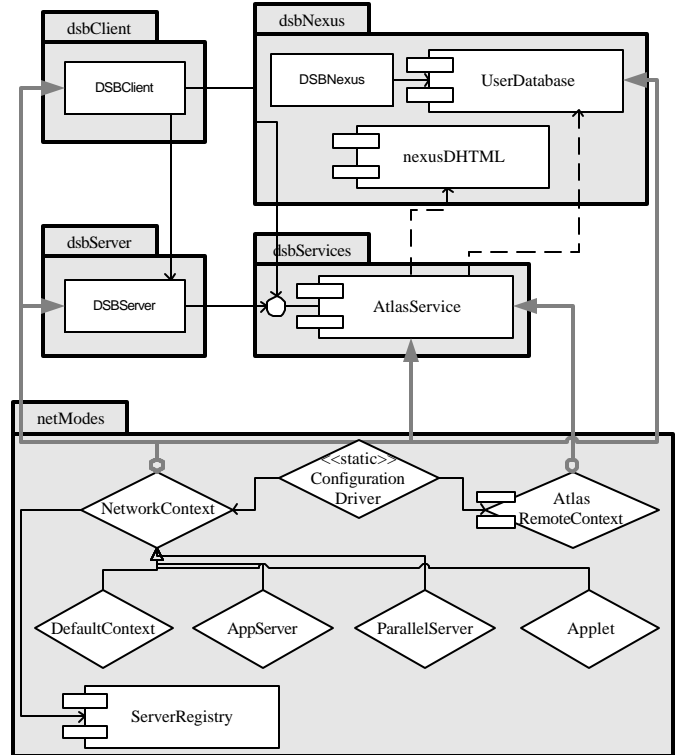


Figure 2 Aspects in the Network Contexts

distributed application contexts to run correctly. We encapsulated all of the distribution-safe behavior within the AtlasRemoteContext aspects. These aspects override calls to context-sensitive objects, such as file streams, to enable them to run in a distributed context.

The NetworkContext aspect and its sub-aspects are dynamic aspects; this permits a context to be created and destroyed at run-time, making possible dynamic reconfigurations. For example, if an unusually high bandwidth was recognized while the DSB was running in single server mode, the applet aspect could be instantiated and used by that client.

Aspects have made it possible to separate the code of the simple single server Atlas system from the other, more complicated, configurations. The separation has been helpful in debugging: support for a particular context may be added or removed, facilitating the isolation of faults. This separation also makes it easier to read through the code base; a developer can build an understanding of the basic Atlas functionality before tackling the issues of distribution and the various configurations.

SUMMARY

Aspects made it possible to separate the code of the simple single server Atlas system from the other, more complicated, configurations. The separation eased debugging as support for a particular context can be added or removed, facilitating the isolation of faults. This separation also makes it easier to read through the code base; a developer can build an understanding of

³ An arrow from an aspect to a package indicates that the aspect acts on the source code comprising the package. More detail about the graphical notation used is provided in Section 4.3.

the basic Atlas functionality before tackling the issues of distribution and the various configurations.

ACKNOWLEDGEMENTS

George Tsiknis and Ian Cavers helped design Atlas from the perspective of a web-based course tool. We thank Rob Walker, Martin Robillard, and Gregor Kiczales for commenting on others drafts of the work reported in this paper. Liz Kendall provided insight on the aspect model notation.

REFERENCES

[1] Kersten, M. and Murphy, G.C. Atlas: A Case Study in Building a Web-based Learning Environment using Aspect-

oriented Programming. In *Proc. of OOPSLA*, 1999, p. 340-352.

[2] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. Aspect-oriented programming. In *Proceedings of ECOOP'97* (Jyväskylä Finland, June 1997), Springer Verlag, 220-242.

[3] Lopes, C. and Kiczales, G. Recent Developments in AspectJ™. In *ECOOP '98 Workshop Reader*, Springer Verlag, 1998, 398-401.